Algorithmics 3 Assessed Exercise

Status and Implementation Reports

Anton Belev
1103816

November 17, 2013

Status report

I believe that my programs are running correctly. There should not be any problems with compaling them. I have tested them against different data sets and the results from the programs were looking correct. The example from the exercise specification document gives the following output against the first program:

```
The shortest path from flour to bread is with length 6
flour->floor->flood->blood->brood->broad->bread
```

And against the Dijkstra's algorithm:

```
The shortest path from flour to bread is with weight 54
flour->floor->flood->blood->brood->broad->bread
```

Another reason to think that the programs are correct is because, I have compared my results with other guys from the course and these results were the same, which makes me feel that I am on the right track.

Finally here is the results I have obtained against the test you have provied for us.

Results from the first program

```
»>
»>
print paint
Total lenght 1
print->paint

Elapsed time: 299 milliseconds
```

```
»>
»>
forty fifty
Total lenght 4
forty->forth->firth->fifth->fifty

    Elapsed time: 314 milliseconds
»>
»>
cheat solve
Total lenght 13
cheat->chert->chart->charm->chasm->chase->cease->lease->leave->heave->helve-
>halve->salve->solve

    Elapsed time: 326 milliseconds
»>
»>
worry happy
No ladder is possible!

    Elapsed time: 296 milliseconds
»>
»>
smile frown
Total lenght 12
smile->smite->spite->spice->slice->slick->click->clock->crock->crook->croon-
>crown->frown

    Elapsed time: 332 milliseconds
»>
»>

    small large
Total lenght 16
small->shall->shale->share->shard->chard->charm->chasm->chase->cease->tease-
>terse->verse->verge->merge->marge->large

    Elapsed time: 332 milliseconds
»>
»>

    black white
Total lenght 8
black->blank->blink->brink->brine->trine->thine->whine->white

    Elapsed time: 337 milliseconds
```

```
»>
»>

    greed money
No ladder is possible!

    Elapsed time: 309 milliseconds
»>
»>
```

Results from the second program

```
»>
»>
blare blase
Total weight 1
blare->blase

    Elapsed time: 318 milliseconds
»>
»>

    blond blood
Total weight 1
blond->blood

    Elapsed time: 351 milliseconds
»>
»>

    allow alloy
Total weight 2
allow->alloy

    Elapsed time: 309 milliseconds
»>
»>

    cheat solve
Total weight 96
cheat->chert->chart->charm->chasm->chase->cease->lease->leave->heave->helve-
>halve->salve->solve

    Elapsed time: 343 milliseconds
```

```
»>
»>

    worry happy
No ladder is possible!

    Elapsed time: 344 milliseconds
»>
»>

    print paint
Total weight 17
print->paint

    Elapsed time: 338 milliseconds
»>
»>

    small large
Total weight 118
small->shall->shale->share->shard->chard->charm->chasm->chase->cease->tease-
>terse->verse->verge->merge->marge->large

    Elapsed time: 354 milliseconds
»>
»>

    black white
Total weight 56
black->slack->shack->shank->thank->thane->thine->whine->white

    Elapsed time: 318 milliseconds
»>
»>

    greed money
No ladder is possible!

    Elapsed time: 374 milliseconds
»>
»>
```

Implementation report

(a) My implementation of the Dijkstra′s is pretty simple. The structure is basically similar to BFS, but there is an additional weight field to the AdjListNode class. Another important difference is that instead of queue, I′m using priority queue so that the next vertex that should be visited will have the smallest weight difference in respect to the current vertex. That way I can obtain the path with the minimum weight. Implementing the algorithm with priority queue is more efficient, than using for example linked list, because it gives constant time access to the next word with highest priority. In our case the highest priority will be given to the word which has an edge with the smallest weight to the current vertex. The overall complexity of the algorithm should be O$(E \times \log V)$ where E is the number of edges and V is the number of vertices.

(b) In order to obtain the total weighted path I have used the distance field from the Vertex class. In the first part of the assignment when the next vertex is visited this distance field is incremented by one. In the Dijkstra′s algorithm I′m incrementing this field not by one but by the weight of the edge. This allows me to get the total weighted path in constant time when I reach the final vertex. Another thing that I need is a predecessor index. When I reach the final vertex I′m using this index to get the previous vertex during the traversal. Finally in order to print the vertices in the correct order, not in the reverse, I′m using a stack.

Empirical results

Depending on the processes that are currently running on my PC the executing times vary between 300-450 milliseconds.